

Software Life-cycle and Integration Issues for CS&E R&D Software and Experiences from Trilinos (Part I)

Roscoe A. Bartlett

http://www.cs.sandia.gov/~rabartl/

Department of Optimization & Uncertainty Estimation Trilinos Software Engineering Technologies and Integration Lead

Sandia National Laboratories

SIAM Parallel Processing 2010, 2/24/2010



Overview of CS&E Software Engineering Challenge

- Progress in Computational Science and Engineering (CS&E) is occurring due to greater numbers of more complex algorithms and methods
 - **Discretization**: a) meshing, b) advanced discretizations, c) adaptively, ...
 - Parallelization: a) parallel support, b) load balancing, ...
 - General numerics: a) automatic differentiation, ...
 - Solvers: a) linear-algebra, b) linear solvers, c) preconditioners, d) nonlinear solvers,
 e) time integration, ...
 - Analysis capabilities: a) error-estimation, b) stability analysis and bifurcation, c) optimization, d) UQ, …
 - New architectures: a) multi-core, b) GPUs, ...
 - Visualization

- ...

- Each technology requires specialized PhD-level expertise
- Almost all technologies need to be integrated into single applications
- Set of algorithms/software is too large for any single organization to create
- Too large to be developed under single blanket of Continuous Integration (CI)

Software Engineering and Software Integration are key bottlenecks for CS&E to have the fullest impact!



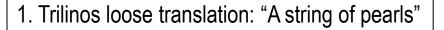


Overview of Trilinos





- Trilinos¹ is an evolving framework to address these challenges:
 - Follow a **TOOLKIT** approach.
 - Fundamental atomic unit is a *package*.
 - Includes core set of vector, graph and matrix classes (Epetra/Tpetra packages).
 - Provides a common abstract solver API (Thyra package).
 - Provides a ready-made package infrastructure:
 - Source code management (git [New]).
 - Build tools (CMake [New]).
 - Automated regression testing (CTest/CDash [New]).
 - Communication tools (Mailman mail lists).
 - Specifies requirements and suggested practices for package SQA.
- In general allows us to categorize efforts:
 - Efforts best done at the Trilinos framework level (useful to most or all packages).
 - Efforts best done at a package level (peculiar or important to a package).
 - Allows package developers to focus only on things that are unique to their package.





Trilinos Package Summary

	Objective	Package(s)										
Discontinutions	Meshing & Spatial Discretizations	phdMesh, Intrepid, Phalanx, Shards, Pamgen, Sundance, FEI, STK, Mesquite										
Discretizations	Time Integration	Rythmos										
NA - Us - sta	Automatic Differentiation	Sacado										
Methods	Mortar Methods	Moertel										
	Linear algebra objects	Epetra, Jpetra, Tpetra										
	Abstract interfaces	Thyra, Stratimikos, RTOp										
Core	Load Balancing	Zoltan, Isorropia										
	"Skins"	PyTrilinos, WebTrilinos, ForTrilinos, Ctrilinos										
	C++ utilities, I/O, thread API	Teuchos, EpetraExt, Kokkos, Triutils, TPI, Optika										
	Iterative (Krylov) linear solvers	AztecOO, Belos, Komplex, Lyno										
	Direct sparse linear solvers	Amesos										
	Direct dense linear solvers	Epetra, Teuchos, Pliris										
	Iterative eigenvalue solvers	Anasazi										
Oshuara	ILU-type preconditioners	AztecOO, IFPACK, Tifpack										
Solvers	Multilevel preconditioners	ML, CLAPS										
	Block preconditioners	Meros, Teko										
	Nonlinear system solvers	NOX, LOCA, Stalix										
	Optimization	MOOCHO, Aristos, GlobiPack, OptiPack, TriKota, Piro										
	Stochastic PDEs	Stokhos, Stalix										

Number of Trilinos Packages:

- Current = 57
- Growth = 5-10 new packages per year!

Number of Trilinos External Third Partly Libraries (TPLs) = 42





- Scalable Computations: As problem size and processor counts increase, the cost of the computation will remain nearly fixed.
- Hardened Computations: Never fail unless problem essentially intractable, in which case we diagnose and inform the user why the problem fails and provide a reliable measure of error.
- **Full Vertical Coverage:** Provide leading edge enabling technologies through the entire technical application software stack: from problem construction, solution, analysis and optimization.
- Grand Universal Interoperability: All Trilinos packages will be interoperable, so that any combination of solver packages that makes sense algorithmically will be possible within Trilinos.
- Universal Accessibility: All Trilinos capabilities will be available to users of major computing environments: C++, Fortran, Python and the Web, and from the desktop to the latest scalable systems.
- Universal Solver RAS: Trilinos will be:
 - <u>Reliable</u>: Leading edge hardened, scalable solutions for each of these applications
 - Available: Integrated into every major application at Sandia
 - <u>Serviceable</u>: Easy to maintain and upgrade within the application environment.

Algorithmic Goals





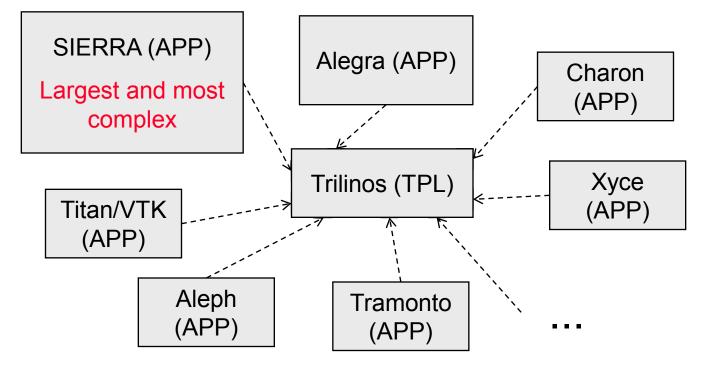
Automatic Intra-Package Dependency Handling

aningm	701 2	92 m	03 T	04 (T	83 :	ree	107 2	rea r	ree r	10 7	11 713	1 713	714 0	F13 F1	10 717	713	216	110 11	1 1 1 1 1	111	129		10 727	123	126 1	36 13	1 732	123	114	733 7.	20 123	11 123	1 238	La co la	r43 (F	42 243	i hee	[ru]		. 6.		1				1.11	1.40 10	Taningai
1) Tolings Transmission	z .	F	TE	Ē	-	-		F	T	Ē	-	F	FF	-	F	FI		ŀ	F	F	F	1	TE	F	FF	-	F	FI		Ē	TE	F	F	F	F	F	F	-	-	TE	F	-	TE	TE	F	1	FF	T01) Tribestfram
12) Teuches	. 12	TT.	1	10	-i	- î				1	1	1						-	1	IF-		1		1		1	1			1		1	1	Ē		-	I.		- 6		1			1F	1	1		T02) Tevakes
(1) Thread Total		F		-6	-				-1	-	1	Ē	Ē		1				-6-	í		-	1	1	i i i	1	1			1	-6	10	1			-6	i –	Ē		1	1			50	÷	1		(703) Thread Total
H) ETDe	- 1		Ē	-6	÷	-	-6	-6	-F	÷	-	E	-	-				-		Ē		-	1	H	H	-	-		-	-			-	÷			÷	-		-F	-			-1-	-	-	E-F	T04) 8.70p
2) Kelden				-6	-	-			-F	-	+	1	-	-			-	÷		÷		-F	-1-	-		-	1		-	÷		-1-	1	÷÷	-		1	-		÷	-			÷	-	÷	H	T03) Kelalara
	-		파는	-15		1	_ <u></u> }		1	-	-			<u> </u>	1		-	<u></u>	1			_ <u>-</u>				-	1		-	-	-15		1	÷	1	_	<u>}</u>	<u>i </u>			-			1	1	<u> </u>		T00) Epsins
(0) Spains	·	_ <u> </u> E		_1	-	_	-		-	1				<u> </u>				<u> </u>	<u>_</u>			1	1		-	-	1		<u> </u>	·····	_	1	1	ĿĿ	_	_ <u>t</u> _	<u>_</u>	<u> </u>	<u> </u>		1			1	1	Ŀ		and the second s
7) Zelten		_ <u>i</u> Ł	<u>_</u>	_ŀ			E -	_1		<u> -</u>		1	<u> </u>		_		_		10	1		_6	<u> </u>		-		1		<u> </u>	÷	1	1	<u> </u>	ĿĿ		_12	<u> </u>	<u> </u>	<u> </u>	_	1	-		_E	1	· .	6	TOT) Zolian
13) Sharela		лĿ	١Ŀ	Ŀ	_	. I.	E	5. I-		1		Ŀ	-	 I 		Ŀ	-	. I.	1		-	- I.	1	Ł	Ł. Ł	1	Ŀ			_ ŀ.	Ŀ		1	ĿĿ	<u> </u>	- E		· 1	- E	Ŀ	1		- E	E	E.		E	(193) Sharein
00) CishiFash	- 5	z	٦Ŀ	E	1	- I-		- F	ε.			Æ		. I+			-	· •	1	ŀ	-	- 12					- E		- I+		Jł.	- E	1	ŀ		- 6	ŀ	· 1	-	ŀ				TE	÷.	•	E	T00) CiebiTash
(B) Triville		TE	٦F	٦F				E					FF		1				1			-		F	FF			F			TE	F		1	F	1.0		1		T	F	- I		TE	F		FF	F10) Triutile
11) Tpains				- F			TE	F	TE	2		F	FF				-		1	Ē	F	-	TE	E	FF	-	1	FI			TE	1	1E	· ·	F		1	F		TE	F		TE	TE	F	ŀ	FF	(T11) Tpairs
12) Epsins Ext		20		-6				-i	-		1			-			-i	-	10	i		-		Ē		-	1			-	-	1	1	i i		-6	i –	i i		10	-			1 C	1	1		(712) Epsira Dat
3) Thoma	-	tional Real		= h	_			-6		10	100	i E		-			1	-	-6	ïH		-	1	1	-h	-	1		-	-		1	1	÷	-6	-6	÷	-			-			16	1	-		F13) Thoma
(4) OptiFash						1.0						1	-	-			-	-		H		-	-	-		-	-		-	-			-	÷÷			÷	-			-			-12	+	-	H	T11) Colifash
	_		-P	<u> </u>	_	_	سل استعب	-12	_		_	a haard	-	÷				-		÷		-ŀ	-ŀ-	H		- <u>-</u> -	-		-	-		-	-	÷÷	-		÷	÷	-	-ŀ-	-			-1-	+	÷	÷	and an other states and the states a
13) las mepis		z Ŀ	<u>_</u>	_1	_	1 2	5. L	<u> </u>	E	uo .	100	1	E_P	× -				<u>.</u>	<u>_</u>	Ŀ		_ <u>ŀ</u>	<u> </u>	1	<u>L</u> Ł		1		<u> </u>	_ <u> </u>	_ <u>L</u>	1	1	ĿĿ	1	_	Ŀ	<u>i </u>	<u>- t</u>	<u>_</u>	Ł	-	L	JŁ.	.t.,	Ŀ	L	(713) Denoração
O) Ties	· ·	_k	리는	ŀ.	_		lt	E	E	Ŀ	1	Ł		2			_	. S. I.	123	Ŀ	E. 1	E	Ŀ	E.	ŁŁ		1		<u> </u>	_ ŀ		Ŀ	1	ĿĿ	<u> </u>	_ 22	Ŀ	Ŀ	· .	٦Ŀ	1			Ŀ	Ł.	·	tt	(716) Firm
T) Clage		1	E	10							1							<u>.</u>	1			1.	F		1					- I-				I. I.		1		1. 1		F	1			E	1	1		717) Claga
OCierta (E		-	TF	TF			F		F			F	F		F	=		1	F	IF 1	F	1	TE	E	FF		F	FI			TE	F	F	1		E	1F	-		TE	F		TE	1F	E		FF	(11) Adm00
P) Calari	-			10				- i	E	- 92	100						- 1	-			FI	-	F	F	FF	-	1			-	1	1	1	L I				1		F	F		TE	TE	F		FIL	F19) Cales
10) Amaza	-	_		-it	_					10	10	_		-				-	1	i -		1	1			1	1			- i-			1	1		1	1	1			1			10	T	1		710) Amazo
(1) Farryan		-6		=6	-	-		=F	-6	-	-	1	1	-				12		í		-		1	H	-	1	-	-	-			1	÷			1				-			-6	-	1	H	(733) Faingan
		=F	=	=F	-	-				-F	1	1	-					_		1		-	1	-	1	-	-		-				1	÷			1	-			-				-	-		
1) Dipash				-#		1.2		-1-		10.		-		·	-	60		.e .	1		-	-ŀ	1	-	1	-	-		-	·	-1-	-	-	÷	-	_	1	÷	-	-1-	-			-1-	-	ŀ.		711) Ngash
2) Remples				_1		<u>بع</u>	1	1		12.	-			· ·	1	1.2		·	-	E			1		L	-	1		-	·	_	1	-	<u>ا ا ا</u>	_	_		<u>i</u>	-	1	1			4	-	Ŀ	L	(713) Stemples
4) ML				_	-			1	E	10	100	E		uo .		20			60	ŀ	=	ŀ	Ŀ	E	ŁŁ		jł		E.	<u>ः</u> ।		E.		ĿĿ		_E.	1	ŀ.	- E	Ŀ	F.			JŁ.	F.	ŀ	E.E	TT4) MIL
3) Deles		-		Ē			E		E	1	•		EF		E		· F		E	E	E	ε.	E		EF		E.			1	DE	E	E	F F		16	E	· 1		E	F			1E	F	ŀ	E	723) Beins
() Tilpenin		z Fr		F		10				-	2		F			FI			F		FI	z	F	E	FF		1						1	1 I			1	. 1			F				F		FIL	(T10) Tilgenia
1) Institution			_	_	_		100		E	_		2.5	FE	- 0.0		20			100				E	1	FF	-	F	FI		- F		1	1	E E			1	1		T	1		TE	TE	F	-		(727) Biratimikes
I) Mares					_		مر (است	-F	_	10 .	L.R.	a haard		-				-	-			-		E	-f	-	1		-	-			1	t		-			-h		1			1	1	1	H-F	T13) Maren
P) 751			-12	-		_					12.0		H			10		-		÷	-	-b		i i i i i i	E-f	÷	1	H	-	-			-	÷÷			1	-			-			-1-	-	-	F-F	T10) FEI
		and the		<u>_</u>	_		and in		and the second		-	a lange									-	_				-	-		-	1	1	1	-	÷	1	_6	Ŀ	-	<u> </u>		1			4	+	Ŀ		
(0) Amanani					_	-0	_	_1	and the second second		0 1120	and the state of t	E-F	<u> </u>	_ <u>L_</u>			<u> </u>		Ŀ	1	_ <u>t</u>	<u> </u>	L	<u> </u>	· ·	1		-	· ·		<u> </u>	<u>.</u>	ĿĿ	<u> </u>	_	Ŀ	<u>i i</u>	<u> </u>	_ <u>L</u>	Ł.	-		_L_	<u>k</u> .	ŀ	ELF	T30) Anasad
1) E.SCen	. 5		10 13	.0	10	1.2	20	_	E	10 11	10 10	:10	1	<u> </u>	<u>_</u>			201	100	1 <u> </u>	1				- 1-	2 X	1			<u></u>	_6	1	1	<u> </u>		_ 62	1	<u>. </u>	_	1	1	1		JE.	1	I	E.E	(731) 8.0Cm
(3) Trilleia		π.				1.2	10		E	10 /	22	2.2	-	1	-	-						-		E	F		1		- 1			-	1	· ·		- 63	-	· 1			F			- E	1		E	733) Tellieta
12) Bieldes	- 6	2 1					10			10 11	10 12	1.0	F	10		1.0	1.0	10	20		20	1	F	F	FF		10	F			F		1	F F		1.0	1	· 1		TE	F			TE	1	-	FF	733) Simbles
14) STR		x [-	2	-it		-	20 2	-	F	2.8	11.0			- 93		2.2	1.0	20 .	FLO		100	-			ER I	-			-	1			1	i i			1	-		T	1			1	T	1		T34) STK
11) Issuis		a E						= ř	E	10 13	10 10	1Eco	E	10		ELO	LO B	101.	FLO		FLO D	1		1	E		1ELO	20		= 1			1	î î			1	1	10		1			1F	-	Í.		(733) Inmain
10) Interapid			_		_		10					and the second		10				_	100		100	-6					and the second	ELO I			26	-1-	1	÷		-6-	÷	-			1			ii:	-	-		T30) Interplat
	A. (10) - 41 - 51			the second second		_	10 1				10 110			100					100			-		-	_								-	는는			1				-	-		음	÷	-	는는	
(7) Thelens	-	<u> </u>	25	의법	-		-0	-	-15	10 10					-				_		1.0	-				- 0	100	1.0	-	-z -	2		<u>_</u>	<u>i-i</u>	-	_	<u>-</u>	i-i	<u> </u>	_ <u>}_</u>	-			4	-	Ŀ		(737) Thalana
(3) Thirmh	· ·	_12	그는	_1	_	-	_	_	_		_							<u> </u>	_	Ŀ			1		ĿĿ	_			-	<u> </u>	_ <u>k</u>	-	1	Ŀŀ		_	Ł	<u> </u>	<u> </u>	Ŀ	-			11-	<u> </u>	Ŀ	<u>L</u> L	723) Phinmh
10) 7/02			-	-	_			_1	E		10 100	60	Ł.ŧ	100	Ŀ	60			20		10		100	E.			100	1.0	- 1		Ŀ	Ŀ	E.	ĿĿ		_	Ł	Ŀ	ĿĿ	Ŀ	F.		<u> </u>	Ŀ	÷.	. I	ΕĿĿ	136) 2/002
(0) Manufal	. 6	2	E	E		1	10	E	E	1.52	1.0	E	E	10	- E	22	1.0		1.0	1	1.2	- F	E	E	EF		E			1	E	E	E	=	E	16	E	· 1	· •	E	F			E	F	· .	E	T10) Meetal
1) TrillessCauplings						1.0	20		E	10 11	10 10	120		LO .	F	10	10	. 0.	10	1	10		100		- 6	ue .	1.0	1.0		10	T	F	10		=		1				1			15	1		F	F11) TelessCaug
2) Rythman			F	_	_	10				10	-	1.0				F			1		FI	- [·	F	F		-	F							F	F		F	-			F			75	F		FIL	F42) Rythmen
3) NOOCHO						1.0				10 7		-	-	-					-			-		F	-f	-	1		-				1		F	-					1			1	1	1		TI3) NOOCHO
4) Arisina			F					-F		10	LO	al Instatut	-	1.0		5		-	-		-	-F	1	1	H	-	-		-	- F			1	t-F	-		÷	-			-			-1-	-	1	-F	T117 Arbitra
1) Fire		_								_	-					a second second					·	-	-				1						-	÷				-	-		-			금는	-	-		F12) Der
				_		1.2	_		winited by	_	10 1.0	and the second second			1	1.0			1.0	_			11.0			_			_	- 0.2	1	1	10	i li		0 00		=	1	1	1				-	ŀ.		and the second s
0) Jundanm					_	_	10	12	_	-	10 1.2			20	1	1.2	_	_	1.2		1.2		1.2	1	-	2	pro.	10		1.2	2	1	2.2	t_t		20	1	. I	Ξ.	E	1	1		JE.	F.	ŀ	1	T(0) Jundance
T) CTolleas	. 1		1	E		- 2		F		10	12.0		E			-0			-			1	-		ł. F	-	E				E	-		ŀ				· 1	. 1		1	4		E	F			PUT) CTollege
PerTrilees		z .	F	F			20	F	F	10.	11.0		E		E				10	-	E	1	E	E	F		E	E		1	F	E	1	· ·	F	-	F				F			1E	F	ŀ	EF	T13) PerTeless
P) Ty Tellens		0 F				-	10		F			110		10 10	-	20			10	20	20	20 .	100	F	FF		100	1.0	-	10.	F	F	Lo	1	T.		1F			F	F			TE	F		FF	T(P) Ty Telleres
0) Web Tolines			TE	1	-1	1				F	-	1	-		1	F			-	F	-	-	1	1	FF	1	1	F		-	1	1	1	1		-	1				F	E	TE	TE	T.			(730) Web Trilleon
1) Entrates	-	-6	The second	۲ĥ	÷	-	1		- 6	-	-			-				-	-	1		-		-	H	-	1			-			1	i - i	-		1				-			1E	1	1		Tatt Distantion
1) Tex Tankara	-	1	÷È	극는		-				-	-		-	-			-	-	1	÷		-			-	-	1		-	-			-	는분		-	1	-			-		-	12	-	-		T22) Nev Tankers
			-	-14	1	-	-	-	-	÷	-	-		-				-	-		-	-ŀ	-		÷	-	1		-	-	-1-	-	-	i li	-	_	1	Ŀ.	-		-		_F	1	-	<u> </u>		
3) Optiles	- 5	~		_1	_	-	<u> </u>	_	<u> </u>	ŀ	-	E	Ŀŀ	· •	_			· ·					Ŀ	L	E-E	-	E		-	ŀ	-	1	1	Ŀŀ	_		1	Ŀ	· •	E	F.	النظ	E	E	1	ŀ	1	F33) Optitus
4) Magula	•	_E		Ŀ	_	-	1	E	E	ŀ	+	Ł	E.F	1		Ł			E	ŀ	Ł	ŀ	Ŀ	E.	ΕĒ	-	E.	Ł		ŀ	Ŀ	1 L	Ł	ŀŀ		1	Ł	ŀ. I	<u> </u>	. E	1	-		Ŀ	Þ.	ŀ	ΕĿ	734) Magular
2) Siele	. 1	2 1			-		-	E	E	10 10	R 11.0	10	FF	- 92	E	10	10	uo (.	1.0		1.0	ŀ	E	E	F		10	ELO P		un l	E	E	E	l. I.	F	E	E	· 1	· .	E	E		E	E	F	1	E	(723) Stelle
() Take		2 2					10	-	F	1.2. 12		E.R.	FF	- 0.0	T	2.2	100		1.0		1.2	- 0.2	1.2	F	F		1	FI			F	TE						-			-		TE	TE	F		FI	Tat) Take
1) Lone					_			- it		_		_	T I	10					100				1	1	E	2	1		-i			1	1			-1-	1	1			-			100	1	1		Tat) Long
		C		يو است	_	_	197			_	_	تسميل او	the second se		_						- A	1.	10	- C								10	1	a 81	T	10.0	18.									100	- E	

http://trilinos.sandia.gov/packages/interoperability.html



CS&E Environment at Sandia National Labs for Trilinos



TPL: Third Party Lib

- Provides functionality to multiple APPs
- The "Supplier" to the APP

APP: Application

- Delivers end user functionality
- The "Customer" of the TPL

- Sophisticated CS&E applications
 - Discretized PDEs (SIERRA, Alegra, Aleph, Charon)
 - Circuit network models (Xyce)
 - Other types of calculations (Titian/VTK, Tramonto)
- (Massively) parallel MPI (Gordon Bell Winners)
- Almost entirely developed by non-software people
- Wide range of research to production (i.e. from Aleph to SIERRA)





Vision for a Confederation of CS&E Software?



The Vision: A Confederation of CS&E Codes?

- Develop a confederation of trusted, high-quality, reusable, compatible, software packages/components including capabilities for:
 - **Discretization**: a) meshing, b) advanced discretizations, c) adaptively, ...
 - Parallelization: a) parallel support, b) load balancing, ...
 - General numerics: a) automatic differentiation, ...
 - Solvers: a) linear-algebra, b) linear solvers, c) preconditioners, d) nonlinear solvers, e) time integration, ...
 - Analysis capabilities: a) error-estimation, b) stability analysis and bifurcation, c) optimization, d) UQ, …
 - New architectures: a) multi-core, b) GPUs, ...
- Visualization APP1 CS&E Confederation Pkg B Pkg A Pkg A Pkg B Pkg C . . . APP2 Pkg C Pkg B Pkg A Pkg X Pkg Y Pkg Z . . Trilinos itself is a smaller example of this! Pkg X Pkg Y

Challenges for the Reuse of External CSE Software

Many CS&E organizations and individuals are adverse to using externally developed CS&E software!

Using externally developed software can be as risk!

- External software can be hard to learn
- External software may not do what you need
- Upgrades of external software can be risky:
 - Breaks in backward compatibility?
 - Regressions in capability?
- External software may not be well supported
- External software may not be support over long term

What can reduce the risk of depending on external software?

- Develop and maintain strong organizational relationships
- Long term commitment and support (i.e. 10-30 years)
- Apply strong software engineering processes and practices (high quality, low defects, frequent releases, regulated backward compatibility)





Overview of Modern Lean/Agile Software Engineering





- **Software Life-Cycle:** The processes and practices used to design, develop, deliver and ultimately discontinue a software product or suite of software products.
 - Example life-cycle models: Waterfall, Spiral, Evolutionally Prototype, Agile, ...

Agile Software Engineering Methods:

- Agile Manifesto (2001) (Capital 'A' in Agile)
- Founded on long standing wisdom in SE community (40+ years)
- Push back against heavy plan-driven methods (CMM(I))
- Focus on incremental design, development, and delivery (i.e. software life-cycle)
- Close customer focus and interaction and constant feedback
- Example methods: SCRUM, XP (extreme programming)
- Becoming a dominate software engineering approach

Lean Software Engineering Methods:

- Adapted from Lean manufacturing approaches (e.g. the Toyota Production System).
- Focus on optimizing the value chain, small batch sizes, minimize cycle time, automate repetitive tasks, ...
- Hard to distinguish between Lean and Agile ...

References: <u>http://www.cs.sandia.gov/~rabartl/readingList.html</u>



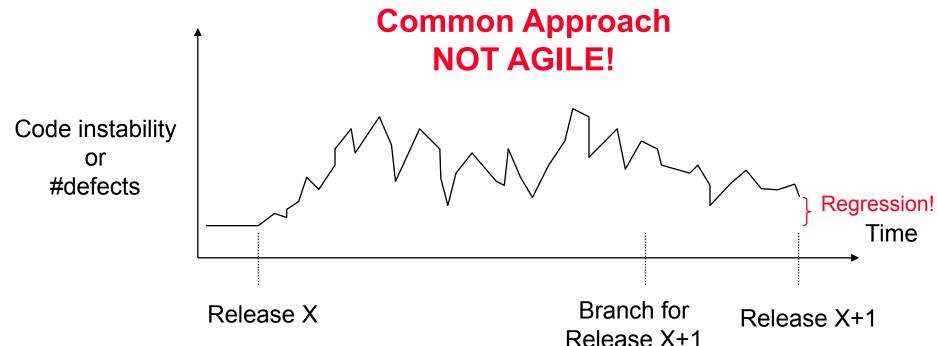
Relevant Lean/Agile Software Engineering Principles

- Agile Design: Reusable software is best designed and developed by incrementally attempting to reuse with new clients and incrementally redesigning and refactoring the software as needed keeping it simple.
 - Technical debt in the code is management through continuous incremental (re)design and refactoring.
- **Agile Quality**: High quality defect-free software is most effectively developed by not putting defects into the software in the first place.
 - High quality software is best developed collaboratively (e.g. pair programming and code reviews).
 - Software is fully verified before it is even written (i.e. Test Driven Development for system verification and unit tests).
 - High quality software is developed in small increments and with sufficient testing in between sets of changes.
- Agile Integration: Software needs to be integrated early and often
- **Agile Delivery**: Software should be delivered to real (or as real as we can make them) customers is short (fixed) intervals.

References: <u>http://www.cs.sandia.gov/~rabartl/readingList.html</u>





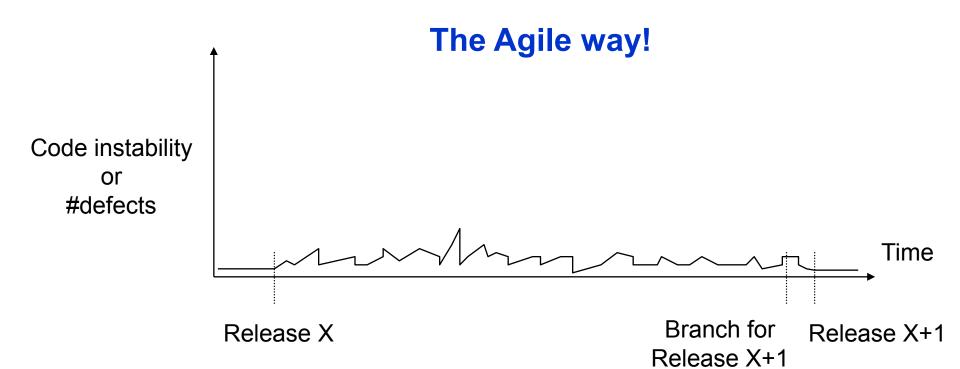


Problems

- · Cost of fixing defects increases the longer they exist in the code
- Difficult to sustain development productivity
- Broken code begets broken code (i.e. broken window phenomenon)
- Long time between branch and release
 - Difficult to merge changes back into main development branch
 - Temptation to add "features" to the release branch before a release
- High risk of creating a regression







Advantages

- Defects are kept out of the code in the first place
- Code is kept in a near releasable state at all times
- Shorten time needed to put out a release
- Allow for more frequent releases
- Reduce risk of creating regressions
- Decrease overall development cost



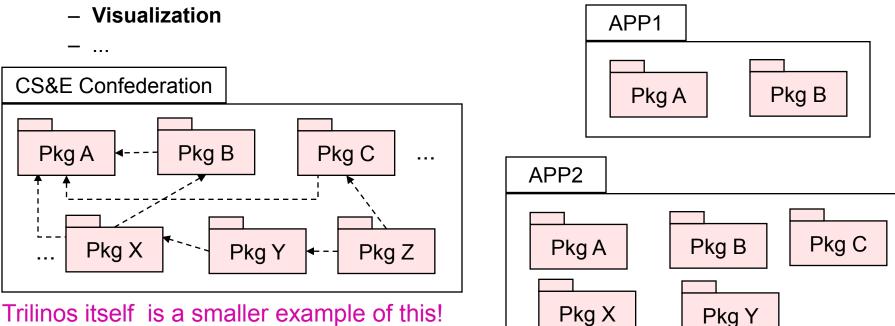


Realizing a Confederation of CS&E Software



The Vision: A Confederation of CS&E Codes?

- Develop a confederation of trusted, high-quality, reusable, compatible, software packages/components including capabilities for:
 - **Discretization**: a) meshing, b) advanced discretizations, c) adaptively, ...
 - Parallelization: a) parallel support, b) load balancing, ...
 - General numerics: a) automatic differentiation, ...
 - Solvers: a) linear-algebra, b) linear solvers, c) preconditioners, d) nonlinear solvers, e) time integration, ...
 - Analysis capabilities: a) error-estimation, b) stability analysis and bifurcation, c) optimization, d) UQ, …
 - New architectures: a) multi-core, b) GPUs, ...



Requirements/Challenges for Confederation of CS&E Codes

Software quality and usability

=> Design, testing, collaborative development

- Building the software in a consistent way and linking
 => Common build approach?
- Reusability and interoperability of software components
 - => Incremental Agile design, resource management, ...
- Documentation, tutorials, user comprehension
 - => SE education, better documentation and examples
- Critical new functionality development
 - => Closer development and integration models
- Upgrading compatible versions of software
 - => Frequent fixed-time releases
- Safe upgrades of software
 - => Regulated backward compatibility, software quality
- Long term maintenance and support
 - = > Stable organizations, stable projects, stable staff
- Self-sustaining software (clean design, clean implementation, well tested with unit tests and system verification tests) => Anyone can maintain it!

The Trilinos is taking (baby) steps to address all of these issues at some level.





Regulated Backward Compatibility



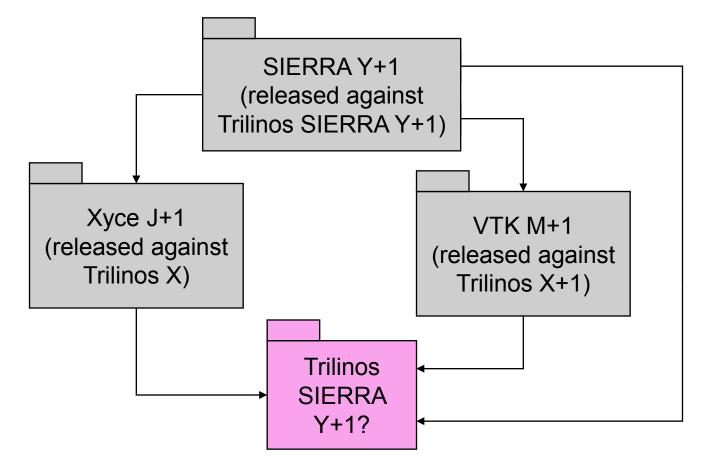


- Backward compatibility is critical for:
 - Safe upgrades of new releases
 - Composability and compatibility of different software collections





Example of the Need for Backward Compatibility



Multiple releases of Trilinos presents a possible problem with complex applications

Solution:

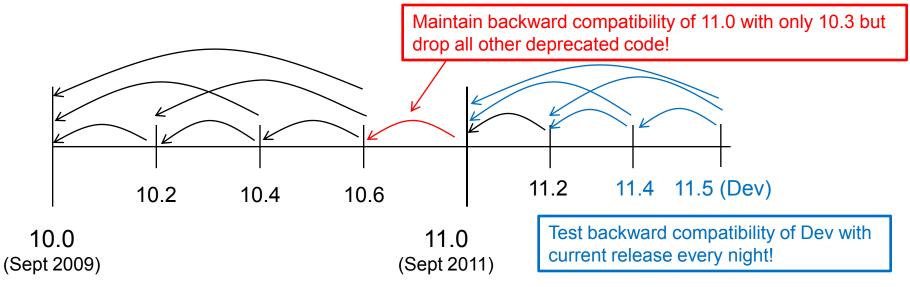
=> Provide sufficient backward compatibility of Trilinos X through Trilinos SIERRA Y+1



- Backward compatibility is critical for:
 - Safe upgrades of new releases
 - Composability and compatibility of different software collections
- Maintaining backward compatibility for all time has downsides:
 - Testing/proving backward compatibility is expensive and costly
 - Encourages not changing (refactoring) existing interfaces etc.
 - => Leads to software "entropy" which kills a software product
- A compromise: Regulated backward compatibility (Trilinos approach)
 - Maintain a window of "sufficient" backward compatibility over major version numbers (e.g. 1-2 years)
 - Provide "Deprecated" compiler warnings
 - Example: GCC's <u>deprecated</u> attribute enabled with _DTrilinos_SHOW_DEPRCATED_WARNINGS:BOOL=ON
 - Provide strong automated testing of Trilinos backward compatibility
 - Drop backward compatibility between major version numbers



- Trilinos Version Numbering X.Y.Z:
 - X: Defines backward compatibility set of releases
 - Y: Major release (off the master branch) number in backward compatible set
 - Z: Minor releases off the release branch X.Y
 - Y and Z: Even numbers = release, odd numbers = dev
 - Makes logic with Trilinos_version.h easier
- Backward comparability between releases
 - Example: Trilinos10.6 is backward compatible with 10.0 through 10.4
 - Example: Trilinos 11.X is not compatible with Trilinos 10.Y



Example: Major Trilinos versions change every 2 years with 2 releases per year



Summary



CS&E Software Engineering Challenge

- Progress in Computational Science and Engineering (CS&E) is occurring due to greater numbers of more complex algorithms and methods
 - **Discretization**: a) meshing, b) advanced discretizations, c) adaptively, ...
 - Parallelization: a) parallel support, b) load balancing, ...
 - General numerics: a) automatic differentiation, ...
 - Solvers: a) linear-algebra, b) linear solvers, c) preconditioners, d) nonlinear solvers,
 e) time integration, ...
 - Analysis capabilities: a) error-estimation, b) stability analysis and bifurcation, c) optimization, d) UQ, …
 - New architectures: a) multi-core, b) GPUs, ...
 - Visualization

- ...

- Each technology requires specialized PhD-level expertise
- Almost all technologies need to be integrated into single applications
- Set of algorithms/software is too large for any single organization to create
- Too large to be developed under single blanket of Continuous Integration (CI)

Software Engineering and Software Integration are key bottlenecks for CS&E to have the fullest impact!



Requirements/Challenges for Confederation of CS&E Codes

- Software quality and usability
 - => Design, testing, collaborative development
- Building the software in a consistent way and linking
 => Common build approach?
- Reusability and interoperability of software components
 - => Incremental Agile design, resource management, ...
- Documentation, tutorials, user comprehension
 - => SE education, better documentation and examples
- Critical new functionality development
 - => Closer development and integration models
- Upgrading compatible versions of software
 - => Frequent fixed-time releases
- Safe upgrades of software
 - => Regulated backward compatibility, software quality
- Long term maintenance and support
 - = > Stable organizations, stable projects, stable staff
- Self-sustaining software (clean design, well tested with unit tests and system verification tests) => Anyone can maintain it!

See Part II, Integration Issues at 10:50 AM Possible topics for Round Table Discussion at 6:00 PM



THE END

